Lab Assignment 3 - Lab 8

Problem Statement

Heart Disease Prediction Using MLP

- Check the dataset for missing values and handle, if any.
- Display input and output features of the dataset.
- Encode non-numeric input attributes using Label Encoder.
- Construct an MLP with configuration 11x128x64x32x1. Use Adam optimizer and appropriate activation functions and train the model.
- Predict heart disease for test data and display confusion matrix, accuracy, recall, precision and F1-score.

Theory

In this project, we develop a Heart Disease Prediction model using a Multi-Layer Perceptron (MLP), a type of feedforward artificial neural network.

The model aims to predict whether a patient has heart disease based on clinical features such as age, cholesterol levels, blood pressure, and more.

The dataset was first preprocessed by checking and handling missing values, and encoding categorical attributes using Label Encoding. Feature scaling was applied to standardize numerical inputs, ensuring the model trains efficiently.

An MLP model with architecture 11x128x64x32x1 was built using TensorFlow and Keras libraries. The network uses ReLU activation functions in hidden layers and Sigmoid activation in the output layer to perform binary classification. The model is optimized using the Adam optimizer and trained with binary cross-entropy loss.

After training, the model's performance was evaluated using metrics such as confusion matrix, accuracy, precision, recall, and F1-score, providing a comprehensive understanding of its prediction capabilities.

Program

The following link contains the step-by-step code snippet from google collab notebook.

https://colab.research.google.com/drive/1IyI3P3cHUKvgSNNPCC281bashSzpdVQu?usp=drive_link

Compiled Code

```
import pandas as pd
```

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, precision_score,
f1_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# 1. Load dataset
df = pd.read_csv('lib/reference_dataset/heart.csv')

# 2. Check missing values
print("\nChecking for missing values...")
print(df.isnull().sum())

# Fill missing values if any
df.fillna(df.select_dtypes(include=[np.number]).mean(), inplace=True)


# 3. Display input and output features
print("\nInput features:")
print(df.columns[:-1].tolist())

print("\nOutput feature:")
print(df.columns[-1])

# 4. Encode non-numeric input attributes
le = LabelEncoder()
```

```
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = le.fit_transform(df[col])

# 5. Split dataset into X and y
X = df.drop(columns=[df.columns[-1]])  # all except last column
y = df[df.columns[-1]]  # target column

# Normalize the input features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 6. Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# 7. Build MLP Model (11x128x64x32x1)
model = Sequential([
    Dense(128, input_dim=11, activation='relu'),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# 8. Train the model
print("\nTraining the model...")
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.1, verbose=1)
```

```
# 9. Predict on test data
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

# 10. Evaluate performance
conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("\n=== Model Evaluation ===")
print("Confusion Matrix:\n", conf_matrix)
print(f"Accuracy: {accuracy:.4f}")
print(f"Recall: {recall:.4f}")
print(f"Precision: {precision:.4f}")
print(f"F1-Score: {f1:.4f}")
```

Output

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn8-heart.py
2025-04-26 17:18:34.411631: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical
rom different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-04-26 17:18:35.787016: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical
rom different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.

Checking for missing values...
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64

Input features:
['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS', 'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope']

Output feature:
HeartDisease
D:\msccsit\nn\Lab Assignment\venv\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_d
dels, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2025-04-26 17:18:38.323015: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compi

Training the model...
Epoch 1/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.7150 - loss: 0.6004 - val_accuracy: 0.8108 - val_loss: 0.4837
Epoch 2/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8796 - loss: 0.3914 - val_accuracy: 0.8108 - val_loss: 0.4337
Epoch 3/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8631 - loss: 0.3313 - val_accuracy: 0.8243 - val_loss: 0.4285
Epoch 4/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8744 - loss: 0.3168 - val_accuracy: 0.7973 - val_loss: 0.4358
```

```
Epoch 50/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step

=== Model Evaluation ===
Confusion Matrix:
 [[66 11]
 [17 90]]
Accuracy: 0.8478
Recall: 0.8411
Precision: 0.8911
F1-Score: 0.8654
(venv)
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
```

Lab Assignment 3 – Lab 9

Problem Statement

Iris Prediction using MLP.

- Check the dataset for missing values and handle, if any.
- Display input and output features of the dataset.
- Encode output attribute using one hot encoder.
- Shuffle the dataset and then count and display number of tuples in each class.
- Normalize input attributes using standard scalar.
- Split dataset into training/validation/test sets in 70:15:15 ratio.
- Construct an MLP with configuration 4x32x16x8x3. Use Adam optimizer and appropriate activation functions and train the model.
- Predict species of Iris flower for test data and display confusion matrix, weighted avg. accuracy, macro & micro recall, macro & micro precision and macro and micro F1-score.

Theory

The objective of this project is to predict the species of Iris flowers using a Multi-Layer Perceptron (MLP) model. The Iris dataset contains 150 samples of flowers with four input features — sepal length, sepal width, petal length, and petal width — and three output classes: Iris-setosa, Iris-versicolor, and Iris-virginica.

The major steps of the project include:

1. Data Preprocessing
    a. The dataset was checked for missing values (none found).
    b. The target attribute "Species" was one-hot encoded using the OneHotEncoder.
    c. Input features were normalized using StandardScaler to ensure all features are on a similar scale.
    d. The dataset was shuffled and split into training, validation, and test sets in a 70:15:15 ratio.
2. Model Construction:
    a. A Multi-Layer Perceptron (MLP) model was constructed with an architecture of 4 input neurons, followed by hidden layers of 32, 16, and 8 neurons respectively, and finally 3 output neurons (one for each species).
    b. The hidden layers used the ReLU activation function and the output layer used the softmax activation function.
    c. The model was optimized using the Adam optimizer and trained using the categorical cross-entropy loss function.
3. Model Evaluation:
    a. After training, predictions were made on the test set.

b. A confusion matrix and a detailed classification report (precision, recall, f1-score) were generated to evaluate the model's performance.

This project demonstrates the application of a simple feedforward neural network for multi-class classification problems and highlights the importance of proper preprocessing and evaluation techniques in machine learning workflows.

Program:

The following link includes the step-by-step ipynb file for project.

https://colab.research.google.com/drive/1kWh26q5LpyJ-bvNev5k-pFcHmya9t-jV?usp=sharing

Compiled Code:

```
import numpy as np

import pandas as pd

import tensorflow as tf

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, classification_report


# Load dataset

df = pd.read_csv('lib/reference_dataset/iris.csv')  # Change the path if needed


# Check missing values

print("\nChecking for missing values...")

print(df.isnull().sum())


# Fill missing values if any

df.fillna(df.select_dtypes(include=['number']).mean(), inplace=True)

# Display input and output features

print("\nInput features:")

print(df.columns[:-1].tolist())
```

```
print("\nOutput feature:")
print(df.columns[-1])

# Display counts for each class
print("Class distribution:\n", df['Species '].value_counts())

# Encode output feature using OneHotEncoder
encoder = OneHotEncoder(sparse_output=False)
y = encoder.fit_transform(df[['Species ']])

# Separate input and output
X = df.drop('Species ', axis=1).values

# Normalize input features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Shuffle and Split data (70:15:15)
X_temp, X_test, y_temp, y_test = train_test_split(X_scaled, y, test_size=0.15, random_state=42,
stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=(0.15/0.85),
random_state=42, stratify=y_temp)

print(f"Training samples: {len(X_train)}")
print(f"Validation samples: {len(X_val)}")
print(f"Test samples: {len(X_test)}")

# Build MLP model
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(4,)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train model
history = model.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(X_val, y_val), verbose=1)

# Evaluate and predict
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Confusion matrix and classification report
print("\nConfusion Matrix:")
print(confusion_matrix(y_true, y_pred_classes))

print("\nClassification Report:")
print(classification_report(y_true, y_pred_classes, digits=4))
```

Output

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn9-iris.py
2025-04-26 17:10:52.470390: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see sligh
rom different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-04-26 17:10:53.827934: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see sligh
rom different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.

Checking for missing values...
SepalLength    0
SepalWidth     0
PetalLength    0
PetalWidth     0
Species        0
dtype: int64

Input features:
['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth']

Output feature:
Species
Class distribution:
 Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
Training samples: 104
Validation samples: 23
Test samples: 23
2025-04-26 17:10:57.066703: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimiz
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow wit
Epoch 1/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.4766 - loss: 1.0570 - val_accuracy: 0.5652 - val_loss: 1.0018
Epoch 2/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.4138 - loss: 1.0439 - val_accuracy: 0.5217 - val_loss: 0.9732
Epoch 3/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.4886 - loss: 0.9909 - val_accuracy: 0.5217 - val_loss: 0.9439
Epoch 4/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.4257 - loss: 0.9834 - val_accuracy: 0.5217 - val_loss: 0.9134
Epoch 5/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5336 - loss: 0.9404 - val_accuracy: 0.6087 - val_loss: 0.8818
Epoch 6/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.4909 - loss: 0.9052 - val_accuracy: 0.5652 - val_loss: 0.8495
Epoch 7/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.4982 - loss: 0.8811 - val_accuracy: 0.6087 - val_loss: 0.8178
Epoch 8/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5486 - loss: 0.8336 - val_accuracy: 0.6522 - val_loss: 0.7861
Epoch 9/50
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9792 - l
Epoch 50/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9692 - l
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step

Confusion Matrix:
[[7 1 0]
 [0 7 1]
 [0 0 7]]

Classification Report:
              precision    recall  f1-score   support

           0     1.0000    0.8750    0.9333         8
           1     0.8750    0.8750    0.8750         8
           2     0.8750    1.0000    0.9333         7

    accuracy                         0.9130        23
   macro avg     0.9167    0.9167    0.9139        23
weighted avg     0.9185    0.9130    0.9130        23

(venv)
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$
```

Lab Assignment

Problem Statement

- Housing Price Prediction
- Check the dataset for missing values and handle, if any.
- Display input and output features of the dataset.
- Encode non-numeric input attributes using label encoder.
- Normalize input and output attributes using standard scalar.
- Split dataset into training/validation/test sets in 70:15:15 ratio.
- Construct an MLP with configuration 12x128x64x32x16x1. Use Adam optimizer and appropriate activation functions and train the model.
- Predict house price for test data.
- Perform inverse transformation of predicted and actual house price.
- Compute and display RMSE, MAE and MAPE.

Theory

The goal of this project is to predict housing prices using a Machine Learning model. The dataset is preprocessed by handling missing values, encoding categorical features using Label Encoding, and normalizing the data with Standard Scaler to ensure features have a mean of 0 and standard deviation of 1.

The dataset is then split into training, validation, and test sets in a 70:15:15 ratio.

A Multi-Layer Perceptron (MLP) with the architecture 12x128x64x32x16x1 is constructed, where hidden layers use the ReLU activation function and the output layer uses a linear activation (since price prediction is a regression task). The Adam optimizer is used to train the model efficiently.

After training, predictions are made on the test set, and an inverse transformation is applied to convert scaled outputs back to actual price values. The model's performance is evaluated using three error metrics: RMSE, MAE, and MAPE.

The formulas for the metrics are:

1. Root Means Square (RMSE):
   a. $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2}$
2. Mean Absolute Error (MAE):
   a. $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}|$
3. $Mean\ Absoulte\ Percentage\ Error\ (MAPE)$:
   a. MAPE = $\frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y_i}}{y_i}\right|$

(Note: MAPE can be undefined or infinite if any true value y_i is zero.)

Program:

The following link includes the step-by-step ipynb file for project.

https://colab.research.google.com/drive/1mdjkwLqPkV5TDgBvn4Nfn5PlzL-b8gSe?usp=sharing

Compiled Code:

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.metrics import mean_squared_error, mean_absolute_error

import tensorflow as tf


# Load your dataset

df = pd.read_csv('lib/reference_dataset/Housing.csv')


# 1. Check for missing values and handle them

print("Missing Values:\n", df.isnull().sum())

df = df.dropna()  # or you can use df.fillna() for imputation


# 2. Display input and output features

print("\nInput Features:\n", df.columns[:-1])

print("\nOutput Feature:\n", df.columns[-1])


# 3. Encode non-numeric input attributes

label_encoders = {}

for column in df.select_dtypes(include=['object']).columns:

    le = LabelEncoder()

    df[column] = le.fit_transform(df[column])

    label_encoders[column] = le
```

```
# 4. Normalize input and output attributes
scaler_X = StandardScaler()
scaler_y = StandardScaler()

X = df.iloc[:, :-1].values  # All columns except last
y = df.iloc[:, -1].values.reshape(-1, 1)  # Last column (target)

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y)

# 5. Split dataset into 70:15:15
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y_scaled, test_size=0.30,
random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print(f"\nTrain set size: {X_train.shape}")
print(f"Validation set size: {X_val.shape}")
print(f"Test set size: {X_test.shape}")

# 6. Construct the MLP model
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(X_train.shape[1],)),  # input layer
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1)  # output layer (no activation)
])
```

```
model.compile(optimizer='adam', loss='mse')

# 7. Train the model
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=100,
    batch_size=32,
    verbose=1
)

# 8. Predict house price for test data
y_pred_scaled = model.predict(X_test)

# 9. Perform inverse transformation
y_pred = scaler_y.inverse_transform(y_pred_scaled)
y_true = scaler_y.inverse_transform(y_test)

# 10. Compute and display RMSE, MAE and MAPE
rmse = np.sqrt(mean_squared_error(y_true, y_pred))
mae = mean_absolute_error(y_true, y_pred)
epsilon = 1e-8  # Small value to prevent division by zero
mape = np.mean(np.abs((y_true - y_pred) / (y_true + epsilon))) * 100

print(f"\nRMSE: {rmse:.2f}")
print(f"MAE: {mae:.2f}")
print(f"MAPE: {mape:.2f}%")
```

Output:

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn10-Housing.py
2025-04-26 17:44:08.602742: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on.
rom different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDN
2025-04-26 17:44:10.012819: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on.
rom different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDN
Missing Values:
 price               0
area                0
bedrooms            0
bathrooms           0
stories             0
mainroad            0
guestroom           0
basement            0
hotwaterheating     0
airconditioning     0
parking             0
prefarea            0
furnishingstatus    0
dtype: int64

Input Features:
 Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea'],
      dtype='object')

Output Feature:
 furnishingstatus

Train set size: (381, 12)
Validation set size: (82, 12)
Test set size: (82, 12)
2025-04-26 17:44:12.615463: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebui
Epoch 1/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - loss: 1.0441 - val_loss: 0.9283
Epoch 2/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.9415 - val_loss: 0.9020
Epoch 3/100
```

```
Epoch 100/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1284 - val_loss: 1.3951
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step

RMSE: 1.02
MAE: 0.80
MAPE: 2873846646.69%
(venv)
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
```